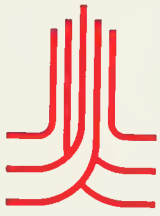
The Power of Multimedia:
Combining Point-to-Point and Multiaccess Networks

by

*Yehuda Afek**
Gad M. Landau†
Baruch Schieber‡
Moti Yung§

Ultracomputer Note #134
Computer Science Technical Report #351
March, 1988



Ultracomputer Research Laboratory

New York University
Courant Institute of Mathematical Sciences
Division of Computer Science
251 Mercer Street, New York, NY 10012

The Power of Multimedia: Combining Point-to-Point and Multiaccess Networks

by

*Yehuda Afek**
Gad M. Landau†
Baruch Schieber‡
Moti Yung§

Ultracomputer Note #134
Computer Science Technical Report #351
March, 1988

ABSTRACT

In this paper we introduce a new network model called a ***multimedia network***. It combines the point-to-point message passing network and the multiaccess channel. To benefit from the combination we design algorithms which consist of two stages: a local stage which utilizes the parallelism of the point-to-point network and a global stage which utilizes the broadcast capability of the multiaccess channel. As a reasonable approach, one wishes to balance the complexities of the two stages by obtaining an efficient partition of the network into $O(\sqrt{n})$ connected components each of radius $O(\sqrt{n})$. To this end we present efficient deterministic and randomized partitioning algorithms. The deterministic algorithm runs in $O(\sqrt{n}\log^* n)$ time and $O(m + n \log n \log^* n)j$ messages, where m and n are the number of links and nodes in the point-to-point part of the network. The randomized algorithm runs in the same time but sends $O(m + n \log^* n)$ messages. The partitioning algorithms are then used to obtain $O(\sqrt{n} \log n$ time deterministic and $O(\sqrt{n} \log n)$ time randomized algorithms, for computing *giobal sensitive* functions and a minimum spanning tree.

An $\Omega(n)$ time lower bounds for computing global sensitive functions in both point-to-point and multiaccess networks, are given, thus showing that the multimedia network is more powerful than both its separate components. Furthermore, we prove an $\Omega(\sqrt{n})$ time lower bound for multimedia networks, thus leaving a small gap between our upper and lower bounds.

---

* AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974.

† The research of this author was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract number DE-AC02 76ER03077. This research was conducted while the author was in residence at the Courant Institute of Mathematical Sciences, New York.

‡ IBM Research Division, T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

§ IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

1 Introduction

Two classes of data networking technologies have emerged in recent years, the point-to-point network (store and forward message passing system), and the multiaccess channel (broadcast channel) [BG87,Tan81]. In the point-to-point message passing network, communication lines connect pairs of processors in an arbitrary topology. In one step, each of the processors in the network can send a message to each of its neighbors. In the multiaccess channel, on the other hand, all processors are connected to a broadcast medium (e.g., bus, Ethernet, satellite, or radio channels). In one step, a single message can be heard by all the processors in the network. When more than one processor tries to access the channel simultaneously, a collision occurs and is detected by the processors. In an attempt to take advantage of both the high bandwidth of the point-to-point network and the broadcast properties of the multiaccess channel, supercomputer designers ([FG87,ABCP87]) and network architects ([KHS88]) have recently suggested to combine these two technologies. (The Intel hyper cube [Int85] is a commercially available system which contains such a combination.) We call a network whose processors are connected by both a point-to-point message passing system and a multiaccess channel a *multimedia network*. In this paper we, first, define the *multimedia network* model, and second, investigate the algorithmic aspects of this model.

Many distributed algorithms have been developed for both point-to-point networks, [GHS83,Gal82,Awe85,Awe87] to mention just a few, and multiaccess networks [Cap79] [Gre82,GL83,Wil84,GM87]. However, to the best of our knowledge, a combination of an arbitrary point-to-point network with a multiaccess network has not been considered yet.

Bokhari [Bok84] and Stout [Sto86] considered the algorithmic aspects of adding a bus system to a mesh connected parallel machine. They showed that the divide-and-conquer approach leads to efficient algorithms in such a combined model. Our algorithms use the same approach. The main subtlety of our algorithms lies in the "divide" stage since, in contrast to the model considered in [Bok84,Sto86], our network is of arbitrary topology.

The main contributions of this paper are: the precise definition and the examination of the power of the multimedia model, and efficient deterministic and randomized algorithms for partitioning the multimedia network.

To benefit from both the parallelism of the point-to-point network and the broadcast capability of the multiaccess channel we apply the divide-and-conquer (local-parallel and global-broadcast) approach to algorithmic design in the multimedia network. To this end, we divide the computation into two stages: a local stage and a global stage. The local stage is carried out in parallel on the point-to-point network. In this stage the broadcast channel is used only for synchronization. The global stage uses the broadcast

channel to combine the partial computations of the local stage. To balance the efforts of the two stages, we partition the network by constructing a spanning forest consisting of $O(\sqrt{n})$ rooted trees each of radius $O(\sqrt{n})$. Both the deterministic and randomized algorithms for constructing the forest run in $O(\sqrt{n} \cdot \log^* n)$ time, thus remaining with in a $\log^* n$ factor from the above balance.

The deterministic partitioning algorithm sends $O(m + n \cdot \log n \cdot \log^* n)$ messages, where m and n are the number of links and nodes in the network. The algorithm constructs the spanning forest and controls the radius of its trees by combining the tree growing techniques of [GHS83] with the symmetry breaking method of the deterministic coin flipping algorithm of Cole and Vishkin as was suggested by Goldberg, Plotkin, and Shanon [CV86,GPS87].

The randomized partitioning algorithm sends $O(m + n \cdot \log^* n)$ messages. Moreover, it is considerably simpler than the deterministic one, and its probabilistic analysis is quite subtle.

To demonstrate our techniques, and the power of the multimedia network, we consider a class of functions called *global sensitive*. Essentially, a function is defined to be *global sensitive* if a change in any of its inputs always changes the output value (e.g. addition). We present a randomized and a deterministic divide-and-conquer algorithms for computing such functions using the forest partition. The resulting deterministic algorithm runs in $O(\sqrt{n} \cdot \log n)$ time and sends $O(m + n \cdot \log n \cdot \log^* n)$ messages. The expected running time of the randomized algorithm is $O(\sqrt{n} \cdot \log^* n)$ and its message complexity is $O(m + n \cdot \log^* n)$.

Three lower bounds on the time required for computing global sensitive functions are presented. In the first two we show that at least $\Omega(n)$ time is necessary in both point-to-point and multiaccess networks. Together with the upper bounds, this implies that *the multimedia network as a whole is more powerful than each of its parts.* The third lower bound is a $\Omega(\sqrt{n})$ time for the multimedia network. Thus, leaving a small gap between the upper and lower bounds.

Finally, we show how to apply the divide-and-conquer approach to derive a deterministic algorithm for constructing a minimum spanning tree in a multimedia network. This algorithm runs in $O(\sqrt{n} \cdot \log n)$ time as well, and sends $O(m + n \cdot \log n \cdot \log^* n)$ messages.

The rest of the paper is organized as follows: The multimedia network model of computation and the complexity measures are presented in Section 2. In Section 3 we show how to transform the multimedia network into a synchronous network and give a deterministic and a randomized technique to estimate n. The deterministic partitioning algorithm is presented in Section 4, and the probabilistic partitioning algorithm in Section 5. In Section 6 we give the algorithms for computing global sensitive functions, and three lower bounds on the time required to compute a global sensitive function

in each of the models. In Section 7 the minimum weight spanning tree algorithm is presented.

2 Model

We consider a set of processors which communicate simultaneously over two media:

1. an arbitrary topology point-to-point network and
2. a broadcast media (a collision bus).

To model the multimedia network we combine the standard model of asynchronous data communication networks with the standard model of a collision bus.

The network part is the point-to-point message-passing model as in [Seg83,GHS83]. The network topology is described by an undirected communication graph $G = (V, E)$, where V is a set of n nodes, representing the processors of the network, and E is a set of m links, representing the bidirectional communication lines operating between neighboring nodes. Messages sent over a link arrive error-free at the other end, after an arbitrary but finite delay.

The network model is combined with a collision bus by connecting all the nodes of the network to a multiple access channel. For ease of presentation the channel is assumed to be slotted. In the sequel it is shown that any unslotted channel can be made slotted if a Frequency Division Multiple Access scheme (FDMA) is available. Every node can write to, and read from each slot on the channel. Each slot is in one of the following three states: *idle*, *success*, or *collision* depending on whether zero, one, or more than one processors write in that slot, respectively. A good account of the multiaccess model and its relation to possible implementation is given in [GL83].

For the multimedia network combination to make sense we equate the performance parameters of its two components. Specifically, the size of a message in the network is assumed to be the same as the number of bits transmitted in one slot. We assume that the size of each message is at most $O(\log n)$ bits plus the size of any data element. For the deterministic algorithms we assume that each processor has a unique id. Although it is not crucial, to make the presentation lucid we also assume that all the ids can be represented in $O(\log n)$ bits. The ids are used in our algorithms in a way similar to [CV86] and [Cap79].

When analyzing the time complexity of an algorithm (and only then) we assume that the message delay and the inter message delay in the point-to-point network is at most one time unit. The length of one slot is assumed to be bounded by a constant number of time units. The communication complexity is the total number of messages sent over the network.

3 Basic tools

Notice that certain problems are easy to solve in the multimedia network. Obviously, broadcasting a message is trivial. Taking a snapshot [CL85] in this model is easy by synchronizing the check point through the bus. Similarly, by assuming that the multiaccess channel is fault free the problem of resetting a dynamic network [AAG87] is also trivialized. Given the conflict resolution algorithms of [Cap79,Gre82,Mol83,GL83,Mol81] and others, the election problem can be solved in $O(\log n)$ time without using the point-to-point network. As a result, a spanning tree can be easily constructed in $O(n)$ time and $O(m)$ messages. Less obvious but not much harder, a synchronizer [Awe85] with $O(1)$ time, and $O(1)$ messages overhead per round can be implemented using the multiaccess channel as shown in the next subsection. Thus, the multimedia model of computation is at least as powerful as the synchronous point-to-point network.

3.1 The channel as a Synchronizer

A *synchronizer* [Awe85] is a mechanism which enables the execution of synchronous algorithms on an asynchronous network. Let us first review the synchronous model of computation.

In the synchronous model of computation all the nodes are connected to a global clock which generates clock pulses ("ticks") to all the nodes at the same time. The time interval between two consecutive pulses of the clock is a *round*. At the beginning of each round, each node decides, according to its state, what messages to send and on which links to send them. Each node then receives any messages sent to it in this round and uses the received messages and its state to decide on its next state. Note that the only knowledge that the clock pulses provide the algorithm is that all the messages of the algorithm sent in that round have been already received.

A synchronizer provides the nodes of an asynchronous network with clock pulses which provide the algorithm at each node with the same knowledge, i.e., that all the messages sent to it in the present simulated round have been already received. Given this knowledge the algorithm can safely proceed to the computation and communication operations of the next simulated round.

The multimedia network is used to implement a synchronizer by first requiring each node to acknowledge the reception of each message of the algorithm on the point-to-point network links. Second, a node is required to transmit a *busy-tone* on the channel as long as not all the messages the node sent has been acknowledged. An idle slot on the channel is then considered as a clock pulse.

Note that the collision channel synchronizer at most doubles the message complexity of the synchronous algorithm, because of the acknowledgments, and multiplies the time

complexity of the algorithm by at most a constant factor.

Corollary 1 *The multimedia network is at least as powerful as the corresponding synchronous point-to-point network.*

Remark: *Slotted from Unslotted FDMA Channel*
A mechanism similar to the channel synchronizer can be used to convert an unslotted channel to a slotted channel assuming first, that an extra channel is available, and second, that an asynchronous idle period can be asynchronously detected by all the nodes. Essentially, one channel is a synchronizer for the other channel. Every node which is active in a slot transmits a busy tone on the extra channel. An idle period on that channel presents the end of the slot and the time to begin the next slot.

We note that these assumptions are practical since FDMA mechanism can provide the extra channel and low level simple hardware can provide the detection of idle periods.

3.2 Multiaccess resolution and estimation techniques

Many resolution techniques, to allocate the channel in the event of collisions, were developed [Cap79,GL83,Mol81,Wil84]. Essentially, these techniques can be viewed as symmetry breaking methods either by comparing the id bits deterministically or by random coin flips. The reader is referred to these papers for details. (We remark that the resolution problem is not a global sensitive function, because it is equivalent to taking the minimum over a *bounded* number of integers. Minimum is global sensitive only over an unbounded range.)

Greenberg and Ladner [GL83] have suggested a randomized algorithm to estimate the number of processors in a multiaccess network. All the nodes start together rounds of coin tosses; at round i each coin has probability $\frac{1}{2^i}$ for "head". A special busy-tone is transmitted by all the nodes which flipped "head". The estimation terminates as soon as there is an idle slot (nobody transmits a busy tone). When it terminates all nodes know k, the number of rounds. 2^k is then, with high probability, a good estimate (up to a multiplicative factor) on the number of processors in the network. This mechanism can be used to implement our randomized algorithms when n is unknown. Similarly, random bits can be used to generate random ids in case those are not given.

The total number of nodes, n, can be estimated deterministically in $O(\sqrt{n} \cdot \log(|id|))$ time, where id is the largest identifier (id) in the network. The detailed description of the scheme is postponed to Section 6. Essentially, we guess the number of nodes in the network and run the deterministic algorithm for computing a global sensitive function, to check if the guess is correct. If the algorithm fails to finish after a corresponding number of iterations, the guess is doubled and checked again, etc.

4 A deterministic partitioning algorithm

Suppose we are given a multimedia network where each of its links is associated a distinct weight. (We note that this is a simplifying but not crucial assumption.) We present a deterministic algorithm for constructing a spanning forest with the following two properties: (1) Each tree in the spanning forest is a subtree of the minimum spanning tree (MST) of the network. (2) The size of each fragment is $\geq \sqrt{n}$ and its radius is $< 8\sqrt{n}$. Our algorithm incorporates the algorithm of [GHS83] for constructing a minimum spanning tree with the techniques of [CV86] as used in [GPS87] for symmetry breaking.

High-level description of the algorithm

The algorithm proceeds in phases. At each phase we maintain a spanning forest such that, each of its trees is a rooted subtree of the minimum spanning tree (MST) of the network. These rooted trees are called *fragments* and, the root of each fragment is called the *core* of the fragment. At phase i, $i = 1, ..., \frac{\log n}{2}$, of the algorithm the size of each such fragment is $\geq 2^i$ and its radius is $\leq 2^{i+3} - 1$. Clearly, after phase $\frac{\log n}{2}$ we will have the desired partitioning. We remark that our algorithm can be slightly modified to work even if n is unknown. See Section 6 for more details.

At each phase, the algorithm performs some inter-fragment computations and some intra-fragment communications. In this respect each fragment can be viewed as a super-node in a network and the algorithm proceeds by inter-node computations and intra-node communication. To control the pace at which fragments grow we use a synchronizer to run the super-nodes in synchrony. The synchronizer is implemented exactly as in Section 3.1. That is, all the nodes will know when each phase is starting (finishing).

We start the algorithm at Phase 0 by taking the spanning forest which consists of each one of the nodes as a core of a fragment of size one. Each Phase i of the algorithm consists of the following seven steps.

Step 1. Each core counts the number of nodes in its fragment, by broadcasting a message out to the leaves and collating it back. Each node v collates to its parent only after receiving a collate from all its children. Node v then collates to its parent 1 + the sum of its children collates.

Let the *level* of a fragment be the integer part of the base two logarithm of the number of nodes in the fragment.

Step 2. Each core whose level equals i computes the minimum weight outgoing link of its fragment by performing another broadcast and collate on the fragment branches. Upon receiving the broadcast message each node first forwards the message to its children and then performs a local search for its minimum weight outgoing link (as in

[GHS83]). Note that since there are at most $2^{i+1} - 1$ links outgoing from each node to the rest of the nodes in its fragment, the local search will take $O(2^{i+1})$ time and messages. Once the local minimum weight was found and a collate from each of the children was received the node collates to its parent with the minimum weight outgoing link of the subtree rooted at the node. The collate propagates in this way up to the core which thus have computed the minimum weight outgoing link of the fragment.

These chosen links define a directed "fragment" graph F as follows. Each vertex in F corresponds to a fragment in our forest. Each edge in F corresponds to a minimum weight outgoing link found in Step 2. That is, if such a link is outgoing from fragment T_1 to fragment T_2, then the corresponding edge in the "fragment" graph is directed from the vertex corresponding to T_1 to the vertex corresponding to T_2. Define the level of a vertex in F as the level of its corresponding fragment. Observe that each connected component of F is in one of the following three forms: (i) A single vertex. Note that the level of this vertex must be at least $i + 1$. (ii) A rooted tree consisting of vertices in level i rooted at a vertex in a level $> i$. (Later we prove that the radius of this root is bounded by 2^{i+3}). (iii) A tree consisting of vertices in level i, with one extra edge. Note that in this case one link was selected by both its incident fragments as the minimum weight outgoing link. We root the tree at the vertex (fragment) with the higher id among these two vertices (fragments) and omit the extra edge outgoing from it. ¿From now on F is considered a forest.

In the next 5 steps we manipulate the "fragment" graph (forest). We describe the operations as if performed between the cores of the fragments (vertices). The message exchange between the cores is carried over the branches of the fragments in the obvious way.

Step 3. Apply Cole and Vishkin [CV86] techniques as suggested in Goldberg, Plotkin and Shannon [GPS87] to color the vertices of F in three colors. Recall that [GPS87] give a parallel algorithm for coloring the vertices of a rooted tree in three colors. Their algorithm runs in $O(\log^* n)$ time using a linear number of processors on an Exclusive Read Exclusive Write Parallel RAM. It can be easily verified that the same algorithm can be implemented also in a multimedia tree network with k nodes in $O(\log^* k)$ time and $O(k \log^* k)$ messages. Applying this algorithm to each of the rooted trees which form F will result in the desired three coloring.

Denote the three colors used to color the vertices of F by **R**, **G** and **B**. Our goal in Steps 4, 5, and 6 is to compute a maximal independent set (MIS) in F. The MIS will be the set of nodes colored with **R**. For this we, first, decrease the number of vertices colored in **B** (steps 4 and 5) and second, replace the color **G** by the color **R** wherever possible (step 6). The implementation of the first step follows the techniques of [GPS87].

Step 4. For each rooted tree T_F in F and each vertex v in T_F, excluding the root

and its children, recolor v with the color of its father. If the root is colored with **R** then recolor each of its children with a color different from **R** and the child's color. Otherwise, recolor the root's children with the root's color and the root with **R**. Note that the coloring is still legal and that the root is always colored **R**.

Step 5. For each vertex v colored in the color **B** check whether all its neighbors are colored in the same color (which may be either **R** or **G**). If this is the case recolor v in the other available color. Note that the coloring is still legal.

Step 6. If vertex v is colored with **G** and all of its neighbors are colored with **B**, then recolor v with **R**.

It is not difficult to see that after Step 6 the set of all vertices colored in **R** is indeed an MIS. Hence, the length of any path between two vertices in F colored in **R** is bounded by three. This enables us to partition each tree in F into subtrees, the radius of each is bounded by four. To this end, we simply break the trees at the **R** colored vertices (except if that vertex is a leaf). We use this partition to update the spanning forest of the network.

Step 7. If vertex v is colored with **R** and v is not a leaf node then remove the edge outgoing from v in F. For each subtree thus created, join the fragments which correspond to its vertices to form a new fragment. The core of each such new fragment is the core of the fragment corresponding to the root of the subtree (which is the unique **R** colored internal vertex of F in the subtree.)

The following claims can be easily verified.

Claim 1 *After Phase i the level of each fragment in the spanning forest of the network is at least $i+1$.*

Proof of Claim By a simple induction. At the beginning of phase i each active fragment is at level i. During phase i each such fragment is combined with at least one more fragment to create a fragment whose size is at least 2^{i+1}. $\in$

Claim 2 *After Phase i the radius of each fragment in the spanning forest of the network is at most $2^{i+4}-1$.*

Proof of Claim This claim is proved also by a simple induction. By the inductive assumption, the radius of each fragment at the beginning of phase i is at most $2^{i+3}-1$. The subtrees created in step 7 have radius at most four with each vertex with radius at most $2^{i+1}-1$ except possibly the root which might have radius at most $2^{i+3}-1$. Thus the radius of the resulting fragments is at most $2^{i+3}-1+3\cdot(2^{i+1}-1) = 14\cdot 2^i - 4 < 2^{i+4}$. $\in$

Time complexity. The time complexity of Steps 1,2 and 7 is $O(2^i)$. The time complexity of Step 3 is $O(2^i \log^* n)$. The time complexity of Steps 4, 5 and 6 is $O(1)$. Summing up for $i = 1, ..., \frac{\log n}{2}$ we arrive at a total time complexity of $O(\sqrt{n} \log^* n)$.

Message complexity. Our message complexity analysis differs from the analysis of [GHS83] only in Step 3. This step contributes $O(n \log^* n)$ messages per phase and a total of $O(n \log n \log^* n)$ messages. Thus the overall message complexity equals $O(m + n \log n \log^* n)$.

5 A randomized partitioning algorithm

In this subsection we present a randomized algorithm for computing a spanning forest consisting of trees each of radius $O(\sqrt{n})$. We prove that the expected number of trees produced by the algorithm is $O(\sqrt{n})$. The running time of the algorithm is $O(\sqrt{n} \log^* n)$ and its message complexity is $O(m + n \log^* n)$. The algorithm can be modified so that it will work when n is unknown and the nodes are anonymous (See Section 3.2.).

Definition: Let $E_0 = 1$ and $E_i = e^{E_{i-1}}$, for $i = 1, ..., \ln^* n$. In words, E_i is given by raising e to the power of e, $i - 1$ times.

We start the algorithm by initializing all the nodes to be in a *free* state. The algorithm proceeds in at most $\ln^* n + 1$ iterations (assume, w.l.o.g. that the radius $> \sqrt{n}$.) Each iteration $i = 0, ..., \ln^* n$ consists of the following four steps.

Step 1. All the nodes are invoked by broadcasting an "awake" message on the broadcast channel.

Step 2. Each *free* node flips a coin which has $MIN(1, \frac{E_i}{\sqrt{n}})$ probability for "head". A node which flipped "head" becomes a *local center.*

Step 3. Each local center computes a connected component by growing a BFS tree to distance at most $4\sqrt{n}$. Each node in the BFS trees is labeled with the distance from the root of its tree. A node which belongs to a BFS tree from previous iterations switches to a new tree only if it reduces its label. Utilizing the synchronizer of section 3 this can be done in $O(\sqrt{n})$ time and $O(m)$ messages (See [Gal82] for details.) To reduce the message complexity of the algorithm, each link which is found to be internal to a BFS tree is removed from the network for the algorithm purposes.

Step 4. All the nodes with label $\leq 2\sqrt{n}$ become *unfree*, all the rest remain *free* for the next iteration. Note that a free node may belong to a BFS tree.

Clearly, the randomized algorithm described above computes a spanning forest consisting of trees each of radius $\leq 4\sqrt{n}$. Note that in the last iteration all free nodes become centers with probability one. Next we prove:

Theorem 2 *The expected number of trees in the spanning forest is* $O(\sqrt{n})$.

Proof Consider an arbitrary partition of the network into connected components, the size of each is $\geq \sqrt{n}$ and the radius of each is $\leq 2\sqrt{n}$. Note that such a partition always exists. We remark that this partition is needed only for the proof and is not actually computed. We call each connected component in this partition a *block*.

To find the expected number of trees we find the expected number of local centers selected in each iteration. For this, let us first find the expected number of free nodes at the start of each iteration. We observe that if a node is free then no node in its block could have been a local center. Clearly, the probability that no local center will be selected in a block in iteration 0 is $\leq (1 - \frac{1}{\sqrt{n}})^{\sqrt{n}} = e^{-1}$ independently of the other blocks. This implies that the expected number of free nodes at the start of iteration 1 is at most $\frac{n}{e}$. In the same way, the probability that no local center will be selected in a block in iteration 1 (given that no local center was selected in iteration 0) is $\leq (1 - \frac{e}{\sqrt{n}})^{\sqrt{n}} = e^{-e}$. This implies that the expected number of free nodes at the start of iteration 2 is at most $\frac{n}{e^{1+e}}$. In general, the probability that no local center will be selected in a block in iteration $i \geq 0$, given that no local center was selected in previous iterations is $\leq (1 - \frac{E_i}{\sqrt{n}})^{\sqrt{n}} = e^{-E_i} = {E_{i+1}}^{-1}$. This implies that the expected number of free nodes at the start of iteration i is at most $\frac{n}{\prod_{j=0}^{i} E_j}$. Recall that the probability for getting a "head" in iteration i is $\frac{E_i}{\sqrt{n}}$. Hence, the expected number of local centers selected at iteration $i \geq 1$ is at most $\frac{\sqrt{n}}{\prod_{j=0}^{i-1} E_j}$. Clearly, the expected number of local centers selected in iteration 0 is $\sqrt{n}$. Summing the expected number of local centers over the $\ln^* n + 1$ iterations gives an expected number of $O(\sqrt{n})$ local centers. Thus, proving the theorem. $\in$

It is not difficult to verify that the time of the randomized algorithm is $O(\sqrt{n} \log^* n)$. (Note that this is the worst case time and not the expected time). To count the number of messages transmitted by the construction of the BFS trees we note that after exchanging a message over a link either the link is added to a BFS tree or is removed from the network for the algorithm purposes. The latter type of message exchanges contributes at most $O(m)$ messages while the former contributes at most $O(n)$ in each iteration, because the algorithm is run in conjunction with a synchronizer. Thus, the total communication complexity of the algorithm is $O(m + n \log^* n)$ messages.

6 Computing global sensitive functions

In this section we define the class of global sensitive functions, give an algorithm for computing these functions in a multimedia network and prove three lower bounds on the time required to compute such functions in each of the models.

Let $S(X, \bullet)$ be a commutative semigroup, where X denotes the semigroup elements set and $\bullet$ is the semigroup operation. Define the function $F_n : X^n \rightarrow X$ in the obvious way. That is, $F_n(x_1, ..., x_n) = x_1 \bullet x_2 \bullet ... \bullet x_n$. We say that F_n is a *global sensitive function* if for each n-tuple $x_1, ..., x_n$, the value of F_n can not be determined by any subset of $n-1$ elements.

Observe that the class of global sensitive functions contains many natural functions. For example: (1) X is the set of integers and $\bullet$ is addition. (2) X is the set of integers and $\bullet$ is the minimum operation. (Note that when the elements set is bounded from below then the function is *not* global sensitive.) (3) $X = \{0, 1\}$ and $\bullet$ is addition modulo two (exclusive or).

The global sensitive functions defined here resemble the notion of the global algorithms of [KMZ84,Awe87] and the global queries of [SR85]. However, our definition is more restrictive.

In this section we consider the problem of computing a global sensitive function when the inputs are distributed among the processors in the network. That is, each one of the n processors is given one element $x_i \in X$ as input. At the end of the computation each processor has to know the value of $F_n(x_1, ..., x_n)$. Given the partition of the network as defined above, we present an $O(\sqrt{n} \log n)$ time deterministic algorithm and an $O(\sqrt{n})$ expected time randomized algorithm for computing F_n in a multimedia network. We also prove an $\Omega(n)$ time lower bound for computing such functions in point-to-point and multi access networks and an $\Omega(\sqrt{n})$ time lower bound for a multimedia network.

6.1 Overview of the algorithms

We divide the computation into two parts a *local* computation and a *global* computation. In the local computation we compute in parallel the value of the function for each of the components defined by the partition. The local computation is done by a simple broadcast and collate along the trees links [Seg83]. This broadcast takes $O(\sqrt{n})$ time and $O(m)$ messages. Upon the completion of the local computation each root has the partial result of the function computed for the inputs in its component. In the *global* computation we broadcast all the partial results on the broadcast channel. The *global* computation is done by scheduling each one of the roots of the trees on the channel. In the deterministic algorithm this scheduling can be done by applying the resolution techniques of [Cap79] in $O(\sqrt{n} \log n)$ time. In the randomized algorithm this can be done in $O(1)$ expected time per root as shown in [MB76], which gives a total of $O(\sqrt{n})$ expected time. (Note that since we have an estimation on the number of the roots it is not necessary to use the sophisticated techniques of [Wil84] for the scheduling.)

We conclude this subsection by describing how the number of nodes, n, can be

estimated deterministically in $O(\sqrt{n}\log(|id|))$ time, where id is the largest id in the network. To do this, we modify the deterministic partitioning algorithm given in Section 4 in the following way. Recall that the partitioning algorithm consists of $\frac{\log n}{2}$ phases. The output of each phase i is a partition of the graph into fragments the radius of each is at most $2^{i+4}-1$. To estimate n, we check at the end of each phase i if the number of fragments is $\leq 2^i$. This is done by applying the resolution technique of Capetanakis [Cap79] for 2^i rounds (which takes $2^i\log(|id|)$ time units) in an attempt to schedule the cores of the fragments on the channel. The algorithm terminates if all cores were scheduled by then, otherwise a new phase is started. After successfully scheduling all the cores, we can find the exact value of n simply by applying the algorithm for computing global sensitive functions. The function to evaluate will be summation and the input instance will consist of one at each node. Clearly, the result of the computation will be n. It is not difficult to see the whole estimation algorithm takes $O(\sqrt{n}\log(|id|))$ time and $O(\sqrt{n}\log^*(|id|)\log n)$ messages.

6.2 The lower bounds

In this subsection we prove lower bounds for computing global sensitive functions in a point-to-point network, a broadcast network and a multimedia network.

Theorem 3 *The computation of a global sensitive function requires $\Omega(n)$ time in a point-to-point network and a broadcast network and $\Omega(\sqrt{n})$ time in a multimedia network.*

Below, we prove the theorem.

The lower bound for a point-to-point network Consider computing a global sensitive function F_n in a point-to-point network which is a path. Let p be a processor in the network which eventually computes F_n. Note that in order for p to compute F_n, messages has to be sent along some path from any vertex and p. The $\Omega(n)$ time lower bound for computing the function follows.

The lower bound for a broadcast network Suppose we are given an algorithm for computing a global sensitive function in a broadcast network. We show that there is at least one input instance for which the algorithm runs in $\Omega(n)$ time. The proof uses adversary arguments. We view the algorithm as a game between the adversary and each of the processors. Initially, each processor knows only its input. As the algorithm proceeds it gets information about other inputs. According to the definition of global sensitive functions at the end of the computation it must have information

which depends on all the inputs. The goal of the adversary is to reveal as few inputs as possible at each step, thus forcing the algorithm to run for a long time. We assume that the adversary has unlimited computation power and that at any step it can fix the inputs in any way which is consistent with previous steps.

We show that an adversary can fix only $2t$ inputs after t steps of the algorithm. That is, the algorithm will broadcast the same messages in the first t steps, no matter what are the values of the rest of the inputs. Note that since the value of the global sensitive function depends on all the n inputs, when the algorithm terminates all the inputs should be fixed. This implies the desired $\Omega(n)$ time lower bound.

Let us show how the adversary can fix at most two inputs at each step t of the algorithm. The adversary partitions the processors into three sets: (i) All the processors which will not try to broadcast at step t. (ii) All the processors with fixed inputs which will try to broadcast at step t given the computation in the first $t-1$ steps of the algorithm. (iii) All the processors which will try to broadcast at step t depending on their input. I.e., the input of each of these processors is not fixed up to step t and for some consistent inputs the processor may try to broadcast. We distinguish between three cases depending on the size of the third set. **Case** A: The third set is empty. Observe that in this case the behavior of the algorithm (i.e., the broadcast) at step t does not depend on any of the unfixed inputs. Hence, the adversary does not fix any additional inputs. **Case** B: The third set consists of a single processor. Observe that in this case the behavior of the algorithm at step t depends only on the input of this single processor. The adversary fixes the input of this processor causing it to broadcast. **Case** C: There are at least two processors in the third set. In this case the adversary fixes the inputs of at most two processors in the set in such a way that they will try to broadcast, causing a collision on the channel. Note that we will have a collision on the channel independent of the values of the rest of the unfixed inputs. Thus, proving our claim.

The lower bound for a multimedia network The lower bound proof for a multimedia network is similar to the lower bound for the broadcast network, given above. Our lower bound proof resembles two other lower bound proofs given for different models: (1) The $\Omega(\sqrt{n})$ time lower bound given in [Bok84,Sto86] for the time needed to add numbers in a linear mesh with a bus, and (2) The $\Omega(\sqrt{n})$ time lower bound given in [VW83] for the time needed to add numbers in a PRAM with one cell of shared memory.

Consider computing a global sensitive function F_n in a multimedia network with the topology of a path. We show that there is at least one input instance for which the algorithm runs in $\Omega(\sqrt{n})$ time. As in the lower bound proof for the broadcast network this proof also uses adversary arguments. The only difference between the two proofs

is that here we have to take into account the information each processor gains using the point-to-point network. Note that after t steps each processor can gain information on the $2t$ inputs of its neighboring processors using the point-to-point network.

We show that an adversary can fix only $4\sum_{j=1}^{t} j$ inputs after t steps of the algorithm. That is, the algorithm will broadcast the same messages in the first t steps, no matter what are the values of the rest of the inputs. Again, since the value of the global sensitive function depends on all the n inputs, when the algorithm terminates all the inputs should be fixed. This implies the desired $\Omega(\sqrt{n})$ time lower bound.

Let us show how the adversary can fix at most $4t$ inputs at each step t of the algorithm. Again, the adversary partitions the processors into three sets: (i) All the processors which will not try to broadcast at step t. (ii) All the processors whose inputs and the inputs of their $2t$ neighboring processors are fixed which will try to broadcast at step t given the broadcasts in the first $t-1$ steps of the algorithm. (iii) All the processors which will try to broadcast at step t depending on their input and the inputs of their $2t$ neighboring processors. (That is, some of these inputs are not fixed.) Again, we distinguish between three cases depending on the size of the third set. **Case** A: The third set is empty. Observe that in this case the behavior of the algorithm (i.e., the broadcast) at step t does not depend on any of the unfixed inputs. Hence, the adversary does not fix any additional inputs. **Case** B: The third set consists of a single processor. Observe that in this case the behavior of the algorithm at step t depends on the inputs of this single processor and its $2t-2$ neighboring processors. The adversary fixes the inputs of this processor and its $2t-2$ neighbors, which are unfixed yet, causing it to broadcast. **Case** C: There are at least two processors in the third set. In this case the adversary fixes the inputs of at most $4t$ inputs in such a way that at least two processors in the set will try to broadcast, causing a collision on the channel. Note that we will have a collision on the channel independent of the values of the rest of the unfixed inputs. Thus, proving our claim.

7 Computing a minimum spanning tree

In this section we present a deterministic algorithm for constructing a minimum spanning tree (MST) in a multimedia network where each of its links is associated a distinct weight. (Again, we note that this simplifying assumption is not crucial). Our algorithm is actually an implementation of the sequential algorithm of [Kru56]. It has three stages. In the first stage we compute a spanning forest using the deterministic partitioning algorithm given in Section 4. Recall that this spanning forest has the following two properties: (1) Each tree in the spanning forest is a rooted subtree of the MST of the network. These trees are called *initial fragments*. (2) The size of each fragment is $\geq \sqrt{n}$ and its radius is $< 8\sqrt{n}$. In the second stage we compute a scheduling of the

roots of these fragments, for accessing the channel. This is done using the resolution technique of [Cap79]. In the third stage we join the initial fragments to get the MST of the network. Below, we describe the third stage of the algorithm.

The third stage has two parts. First, each node finds out which initial fragment is on the other side of each of its incident links. This first part takes $O(1)$ time and $O(m)$ messages. The second part proceeds in phases. The input to each phase is a spanning forest consisting of rooted fragments of the MST. We call these fragments the *current* fragments. In each phase each current fragment computes its minimum weight outgoing link and then, the current fragments are merged along their selected minimum weight outgoing links to form bigger current fragments. The output of each phase is thus another spanning forest with at most half as many current fragments as the input to the phase. The computation is done using the initial fragments computed in the partitioning algorithm and which remain the same throughout this stage. Thus, at each phase each node belongs to some *initial* fragment and to some *current* fragment. Moreover, inductively, in the beginning of each phase every node knows the names of all the initial fragments in its current fragment. Before the first phase each initial fragment is a current fragment. Each phase consists of the following two steps:

Step 1. The nodes of each initial fragment compute, using the point-to-point network, the minimum weight outgoing link from their initial fragment to a node that is not part of their current fragment. This is done by a simple broadcast and collate on the initial fragment. Note, that since each node knows which initial fragment is on the other side of each of its incident links, this step requires no inter-fragment communication.

Step 2. All the cores of the initial fragments broadcast, using the computed schedule on the broadcast channel, the weight of the minimum weight outgoing link that has been found in Step 1. Each core implicitly broadcasts the following information: (i) the id of its *initial* fragment, (ii) the id of its *current* fragment, (iii) the weight of the link, (iv) the id of the current fragment to which the link is incoming, and (v) the ids of the nodes in both sides of this link. To this end only (iii) and (iv) are explicitly broadcasted the rest can be simulated in each node's memory.

Observe that after this information is broadcasted each node can compute *locally* the minimum weight link outgoing from every current fragment. All these links are part of the MST. Hence each node can compute locally all the newly added MST links and, in particular, all the newly added MST links among its links. Note that by adding all these links to the spanning forest the number of current fragments is at least halved.

Complexity. Computing the initial subtrees takes $O(\sqrt{n}\log^* n)$ time and $O(m+ n\log n\log^* n)$ messages using the algorithm given in Section 4. It takes $O(\sqrt{n}\log n)$ time, to schedule the roots on the channel using the techniques of [Cap79]. In each phase the number of current fragments is at least halved; therefore, there are at most $O(\log n)$ phases. Step 1 of each phase takes $O(\sqrt{n})$ time and $O(n)$ messages. Step 2

of each phase takes $O(\sqrt{n})$ time. We conclude that the algorithm runs in $O(\sqrt{n}\log n)$ time and sends $O(m + n \log n \log^* n)$ messages.

Acknowledgments. We are grateful to Don Coppersmith for helpful discussions and remarks.

References

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. of the 28th IEEE Annual Symp. on Foundation of Computer Science*, pages 358–370, October 1987.

[ABCP87] A. Asthana, C. J. Briggs, M. R. Cravatts, and K. Padmanabhan. A high speed multiple pipeline function unit as a building block for parallel architectures. In *Proc. of the International Conf. on Computer Design*, 1987.

[Awe85] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, October 1985.

[Awe87] B. Awerbuch. Linear time distributed algorithms for minimum spanning trees, leader election, counting and related problems. In *Proc. of the 19th Ann. ACM Symp. on Theory of Computing*, pages 230–240, May 1987.

[BG87] D. P. Bertsekas and R. G. Gallager. *Data Networks.* Prantice Hall, 1987.

[Bok84] K. E. Bokhari. Finding maximum on an array processor with a global bus. *IEEE Trans. Computers*, 33:836–840, 1984.

[Cap79] J. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Trans. on Information Theory*, IT-25(5):505–515, September 1979.

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. on Computer Systems*, 3(1):63–75, January 1985.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proc. of the 16th Ann. ACM Symp. on Theory of Computing*, pages 206–219. 1986.

[FG87] P. A. Franaszek and C. J. Georgiou. Multipath hierarchies in interconnection networks. In *Proc. of the International Conference on Supercomputers*, 1987.

[Gal82] R. G. Gallager. *Distributed Minimum Hop Algorithms.* Technical Report LIDS-P-1175, M.I.T. Lab for Information and Decision Systems, January 1982.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5:66–77, January 1983.

[GL83] A. G. Greenberg and R. Ladner. Estimating the multiplicity of conflicts in multiple access channels. In *Proc. of the 24th IEEE Annual Symp. on Foundation of Computer Science*, pages 384–392, October 1983.

[GM87] Y. Gold and S. Moran. Distributed algorithms for constructing the minimum weight spanning tree in broadcast networks. *Distributed Computing*, 2(3):139–148, 1987.

[GPS87] A. V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. In *Proc. of the 19th Ann. ACM Symp. on Theory of Computing*, pages 315–324, May 1987.

[Gre82] A. G. Greenberg. On the time complexity of broadcast communication schemes. In *Proc. of the 14th Ann. ACM Symp. on Theory of Computing*, pages 354–364, May 1982.

[Int85] Intel. Ipsc: intel's personal supercomputer preliminary data sheet. 1985.

[KHS88] Y. J. Kang, J. H. Herzog, and J. Spragins. Fishnet: a distributed architecture for high-performance local computer networks. *IEEE Trans. on Computers*, 37(1):119–123, January 1988.

[KMZ84] E. Korach, S. Moran, and S. Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In *Proc. of the ACM Symp. on Principles of Distributed Computing*, pages 199–207, August 1984.

[Kru56] J. B. Jr. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *AMS*, pages 48–50, 1956.

[MB76] R. M. Metcalfe and D. R. Boggs. Ethernet: distributed packet switching for local computer networks. *Communication of the ACM*, 19(7):395–403, 1976.

[Mol81] M. Molle. *Unification and Extensions of the Multiple Access Communications Problem.* PhD thesis, UCLA, July 1981.

[Mol83] M. Molle. A simulation study of retansmission strategies for the asynchronous virtual time csma protocol. In *Performance 83*, May 1983.

[Seg83] A. Segall. Distributed network protocols. *IEEE Trans. on Information Theory*, IT-29(1), January 1983.

[SR85] L. Shrira and M. Rodeh. *Methodological Construction of Reliable Distributed Algorithms.* Technical Report, Technion, March 1985.

[Sto86] Q. F. Stout. Meshes with multiple buses. In *Proc. of the 27th IEEE Annual Symp. on Foundation of Computer Science*, pages 264–273, October 1986.

[Tan81] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall, 1981.

[VW83] U. Vishkin and A. Wigderson. Trade-offs between depth and width in parallel computation. In *Proc. of the 24th IEEE Annual Symp. on Foundation of Computer Science*, pages 146–153, October 1983.

[Wil84] D. E. Willard. Loglogarithmic protocols for resolving ethernet and semaphor protocols. In *Proc. of the 15th Ann. ACM Symp. on Theory*

of Computing, pages 512–521, May 1984.

NYU COMPSCI TR-351
Afek, Yehuda
The power of multimedia
c.1

DATE DUE	BORROWER'S NAME